Linux on System z

How to use Execute-in-Place Technology with Linux on z/VM March, 2010

Linux Kernel 2.6 – Development stream

SC34-2594-01

Linux on System z

How to use Execute-in-Place Technology with Linux on z/VM March, 2010

Linux Kernel 2.6 – Development stream

SC34-2594-01

Note
Before using this information and the product it supports, read the information in “Notices” on page 39.

Second Edition (March 2010)

This edition applies to the Linux on System z Development stream and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Summary of changes

This edition contains additional information for the November 2008 software drop. SC34-2594 is the Linux on System z Development stream equivalent to SC33-8287, which applies to the Linux on System z October 2005 stream.

Edition 2 updates

This revision contains additional information for the November 2008 software drop.

New information

- “Accessing a DCSS in exclusive-writable mode” on page 5
- “DCSS options” on page 10

Changed Information

- “Task 4: Copying the shared data to the DCSS” on page 17
- “Updating software on an existing DCSS” on page 29

Deleted Information

- None.

This revision also includes maintenance and editorial changes. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

Updates for the November 2008 software drop

This edition contains changes related to the November 2008 software drop. SC34-2594 is the Linux on System z Development stream equivalent to SC33-8287, which applies to the Linux on System z October 2005 stream.

Changes compared to SC33-8287-00 are:

- You can now map a set of DCSSs that form a contiguous address space in memory to a single DCSS device.
- 64-bit Linux instances that run as z/VM guest operating systems of z/VM 5.4 or later now support DCSS devices of up to 512 GB.
- The mem= kernel parameter is no longer required.

This revision also includes maintenance and editorial changes. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

About this publication

This document describes how you can keep down the memory requirements and boost the performance of a virtual Linux® server farm by using a z/VM® discontiguous saved segment (DCSS). DCSSs can be used by Linux instances that run as z/VM guests on an IBM® System z® mainframe.

In this book, System z is taken to include System z10™, System z9®, and zSeries® in 64- and 31-bit mode.

You can find the latest version of this document on developerWorks® at:

`www.ibm.com/developerworks/linux/linux390/documentation_dev.html`

Who should read this document

This document is intended for Linux administrators and system programmers in charge of a virtual Linux server farm that runs under z/VM.

Note

This document is intended for expert users. Be sure you understand the implications of using a DCSS before you attempt to perform the tasks described in this document.

How this document is organized

Chapter 1, “Introduction to DCSS,” on page 1 introduces the concepts of DCSSs and discusses the options you have when setting up a DCSS for your Linux guests.

Chapter 2, “Setting up a DCSS,” on page 11 provides step-by-step procedures for planning and setting up a DCSS. For some of the tasks there are alternate procedures you can use, depending on the options you choose for:

- The location of your DCSS in storage
- The granularity of the data in your DCSS

The introduction to each of these alternate procedures clearly indicates to which option it applies.

Chapter 3, “Making your Linux guests use the DCSS,” on page 27 summarizes what you need to do for each Linux guest to make it use the DCSS, after you have put the DCSS in place.

Chapter 4, “Updating the software on a DCSS,” on page 29 describes how you can update a DCSS that is already in use.

“Scripts used for setting up a DCSS,” on page 31 contains the scripts that are referred to in the main part of this document. You can also find a tarball with these scripts at:

`www.ibm.com/developerworks/linux/linux390/documentation_dev.html`

Where to get more information

For more information about the CP commands and guest storage, refer to *z/VM CP Commands and Utilities Reference*, SC24-6175.

For information about DCSSs refer to *z/VM Saved Segments Planning and Administration*, SC24-6229.

The book numbers for the z/VM Books apply to z/VM 6. For the complete library including other versions, see `www.ibm.com/vm/library/`.

For more information about the DCSS block device driver refer to *Device Drivers, Features, and Commands*, SC33-8411. You can obtain the latest version of this book on developerWorks at:

`www.ibm.com/developerworks/linux/linux390/documentation_dev.html`

For information about the xip mount option of the second extended file system (ext2), see the following documents in your Linux kernel source tree:

- Documentation/filesystems/xip.txt
- Documentation/filesystems/ext2.txt

Other Linux on System z publications

Current versions of the Linux on System z publications can be found at:

`www.ibm.com/developerworks/linux/linux390/documentation_dev.html`

- *Device Drivers, Features, and Commands*, SC33-8411
- *Using the Dump Tools*, SC33-8412
- *How to use FC-attached SCSI devices with Linux on System z*, SC33-8413
- *How to Improve Performance with PAV*, SC33-8414
- *How to use Execute-in-Place Technology with Linux on z/VM*, SC34-2594
- *How to Set up a Terminal Server Environment on z/VM*, SC34-2596
- *Kernel Messages*
- *libica Programmer’s Reference*, SC34-2602

Chapter 1. Introduction to DCSS

This section explains what a z/VM DCSS is and how DCSSs can help to boost the performance of virtual Linux server farms.

All operating system instances that run concurrently on a System z mainframe vie for some of the available physical memory. When multiple operating system instances need the same data, that data might get loaded into memory several times.

In a virtual server farm with similar Linux instances there is often a considerable amount of data that is required by all instances. A DCSS enables z/VM to load such data into a designated segment of physical memory and allow all Linux instances that need the data to share this memory segment.

A major part of the memory required by a Linux server is used for binary application files and for shared library files. The potential for saving memory by sharing application binaries and libraries depends on the respective applications. In some test scenarios, the same resources could support four times the number of Linux guests for a given guest performance after a DCSS was put in place.

Which data you can share

The shared data must be identical across all sharing Linux instances.

- Application files can only be shared by Linux instances that use the same version of an application.
- Shared data must be read-only. To prevent Linux instances from interfering with one another they are not given write permission for the DCSS.
- Applications and libraries that are frequently used by numerous Linux instances are good candidates for sharing.

The following items are not suitable for sharing:

- Files or directories that are being written to.

  Note: If you choose to share entire directories, the subdirectories of a shared directory are also shared. Therefore, you cannot write to a subdirectory of a shared directory.
- Scripts.

  Sharing scripts is ineffective because they are typically interpreted and not executed directly. Scripts include, for example, files with extension .pl, .py, .sh.
- The /etc directory.

  You cannot share Linux instance specific data. The /etc directory holds, for example, instance specific network configuration data, like a Linux instance's IP address.

Where a DCSS can reside

z/VM has some general restrictions for the size and placement of a DCSS. In addition, there must be no overlap between the address range of a DCSS and the memory of any guest that accesses the DCSS. All guests access the DCSS with the same real addresses. There are two alternative placements for a DCSS to avoid conflicts with the guest's addressing:

- The DCSS addresses are set to be above the highest storage of all individual guests
- The DCSS is placed in the address range between the first and the second storage extent (referred to as a *storage gap* in this document)

The *guest storage* is the amount of memory that z/VM presents to a guest operating system as the guest machines's real memory.

General z/VM constraints for DCSSs

The address limit up to which you can address a DCSS depends on the z/VM level and whether you have a 64-bit or 31-bit Linux system. Table 1 summarizes the upper address limits.

Table 1. Upper address limits at which a DCSS can be addressed

Architecture and z/VM level	**Upper address limit for a DCSS**
64-bit with z/VM 5.4 or later	512 GB
64-bit with z/VM 5.3 or earlier	2047 MB
31-bit (for any z/VM level)	1960 MB

The upper address limit of 1960 MB for a DCSS with 31-bit guests is not actually a DCSS restriction but is the effective limit because 31-bit Linux guests require the address range from 1960 megabyte to 2048 megabyte (2 gigabyte) for virtual memory allocations.

z/VM supports a maximum size of 2047 MB for an individual DCSS. Considering that there is always data other than the DCSS that must be placed below 2047 MB, the upper address limit for 31-bit Linux systems and for z/VM 5.3 or earlier rule out a DCSS of this maximum size. For a 64-bit Linux system with z/VM 5.4 or later, 2047 MB is the actual DCSS size limit.

For a 64-bit Linux system with z/VM 5.4 or later, you can accommodate even more than 2047 MB of DCSS space by defining and using multiple DCSSs with disjunct address ranges. Your Linux instances can load these DCSSs independently and access them as separate DCSS devices. If a set of DCSSs form a contiguous address range, you can also map these DCSSs to a single DCSS device on Linux. If the DCSSs in the set are sufficiently large, the DCSS device can exceed 2047 MB.

DCSS above guest storage

Figure 1 on page 3 shows a DCSS and two Linux guests with their storage. The DCSS must be located within the address range defined by the storage size of the largest guest and an upper limit that depends on your z/VM level and whether you have a 64-bit or 31-bit Linux system.

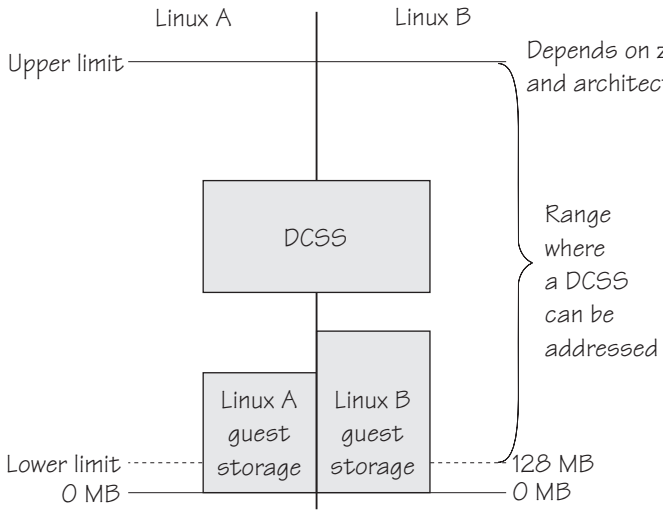

Figure 1. DCSS above Linux guest storage

The upper limits and resulting maximum DCSS sizes are as shown in Table 2.

Table 2. Maximum DCSS size for a DCSS above guest storage

Architecture and z/VM level	**Upper address limit for DCSS**	**Maximum DCSS size**
64-bit with z/VM 5.4 or later	512 GB	2047 MB
64-bit with z/VM 5.3 or earlier	2047 MB	2047 MB – *<largest guest storage size>*
31-bit (for any z/VM level)	1960 MB	1960 MB – *<largest guest storage size>*

For 31-bit Linux systems and for z/VM 5.3 or earlier, the guests' storage must be kept small enough to allow for the DCSS between the storage size of the guest with the largest storage and the 2047 megabyte or 1960 megabyte ceiling. The 512 GB upper limit for 64-bit Linux systems with z/VM 5.4 or later is large enough as not to constitute a restraint for any practical purposes.

Placing a DCSS above guest storage is the preferred option for:

- 64-bit Linux systems with z/VM 5.4 or later
- 31-bit Linux systems

DCSS in a storage gap

You can define your guests' storage as one or more non-contiguous storage extents. Defining non-contiguous storage extents implicitly defines non-addressable storage ranges between the extents. For convenience, this document refers to such non-addressable storage ranges as *storage gaps*.

A DCSS can be placed into an address range that is within the first storage gap for all Linux guests that are to share the DCSS.

Figure 2 on page 4 illustrates a DCSS at an address range that is a storage gap for two 64-bit Linux guests.

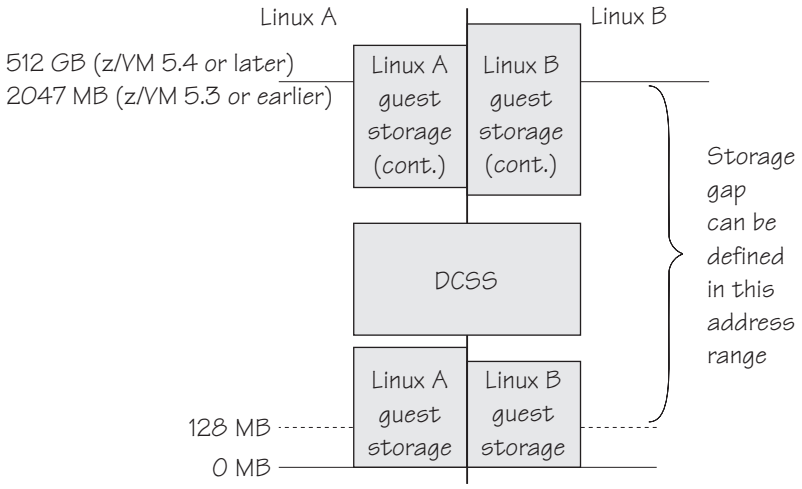

Figure 2. DCSS in a storage gap — 64-bit

To avoid conflicts with data structures that Linux guests require at low memory locations, you are advised to define your storage gap above 128 megabyte. Always use full megabyte boundaries as starting and ending addresses for the storage gap.

For 64-bit Linux guests you can assign guest storage above 2 GB (2048 MB). Locating the DCSS in a storage gap allows a large DCSS to reside below 2 gigabyte without constraining the size of the 64-bit Linux guest storage. For 64-bit Linux guests of z/VM 5.3 or earlier, where the DCSS cannot be addressed above 2047 MB, a storage gap is the preferred DCSS placement. As of z/VM 5.4, you can address DCSSs above 2 GB and so place large DCSSs above guest storage.

As shown in Figure 3, for 31-bit Linux guests both guest storage and the storage gap surrounding the DCSSs must be below 1960 megabyte. A large DCSS, therefore, constrains your guest storage sizes.

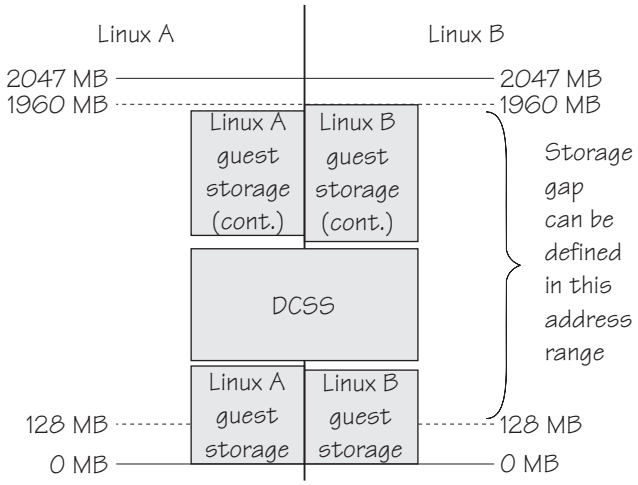

Figure 3. DCSS in a storage gap — 31-bit

Table 3 on page 5 summarizes the maximum sizes for a DCSS in a storage gap.

Table 3. Maximum DCSS size for a DCSS in a storage gap

Architecture and z/VM level	Upper address limit for DCSS	Maximum DCSS size
64-bit with z/VM 5.4 or later	512 GB	2047 MB
64-bit with z/VM 5.3 or earlier	2047 MB	2047 MB – 128 MB = 1919 MB
31-bit (for any z/VM level)	1960 MB	1960 MB – *<largest guest storage size>*

Accessing a DCSS in exclusive-writable mode

You need to access a DCSS in exclusive-writable mode, for example, when creating the initial content or when updating the content of the DCSS.

To access a DCSS in exclusive-writable mode at least one of the following conditions must apply:

- The DCSS fits below the maximum definable address space size of the z/VM guest virtual machine.

  For large read-only DCSS, you can use suitable guest sizes to restrict exclusive-writable access to a specific z/VM guest virtual machine with a sufficient maximum definable address space size.
- The z/VM user directory entry for the z/VM guest virtual machine includes a NAMESAVE statement for the DCSS. See *z/VM CP Planning and Administration*, SC24-6178 for more information about the NAMESAVE statement.
- The DCSS has been defined with the LOADNSHR operand. See “DCSS options” on page 10 about how to save DCSSs with optional properties.

  See *z/VM CP Commands and Utilities Reference*, SC24-6175 for information about the LOADNSHR operand.

How programs run from a DCSS

z/VM allows you to define DCSSs that can be loaded with data and saved on z/VM's spool space. Guest operating systems can load a DCSS into their own address space. If multiple guest operating system instances load the same DCSS, z/VM loads it into memory once only.

While the kernel and all other data that are not shared run from the storage of each guest, shared user space programs are mapped to the DCSS and run directly from the DCSS.

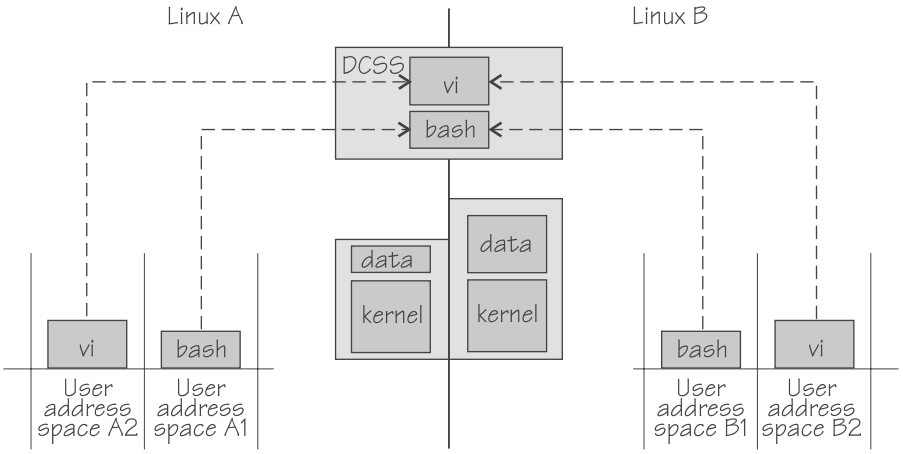

Figure 4. Shared programs run directly from the DCSS

Figure 4 shows a simple example where bash and vi are shared between two Linux guests, Linux A and Linux B.

Linux can access a DCSS using the DCSS block device driver and the second extended file system (ext2) with the xip mount option. To replace directories in the Linux file system with shared content from a DCSS, the shared directories must be over-mounted.

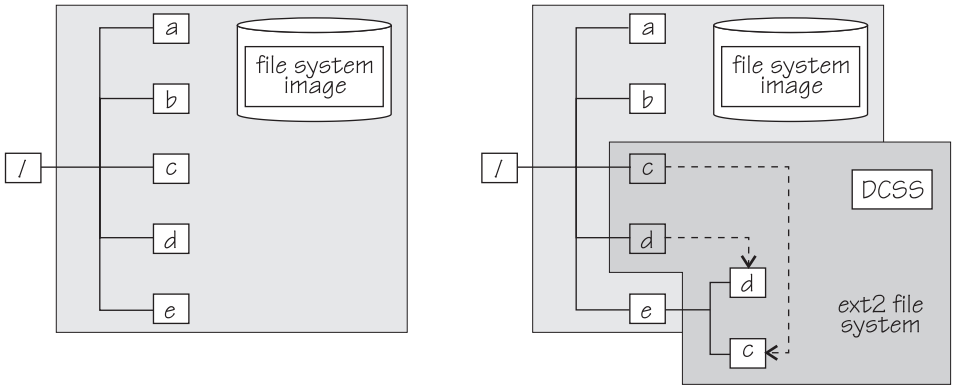

Figure 5. ext2 with the xip mount option over-mounting directories

On the left side, Figure 5 shows a Linux file system with the corresponding DASD volume where it resides as a file system image file. On the right, two of the directories, c and d, have been over-mounted with an ext2 file system. The ext2 file system is mounted on a spare directory, e. Any calls to c are being redirected to e/c and calls to d are redirected to e/d.

Linux loads shared libraries and application files through the mmap() operation. mmap() maps the contents of a file into the application's address space. If mounted with the xip option, the ext2 file system performs this operation by mapping the contents of the DCSS into the application's address space while other file systems use a page cache. The feature enabled by the xip mount option is called execute-in-place because the file contents are not physically copied to a different memory location.

Execute-in-place allows application files and libraries to be accessed without I/O operations. Avoiding I/O operations increases performance. Running applications directly in a shared memory segment saves memory.

Granularity of shared data

A DCSS can contain shared data at the granularity of:
- Entire directories
- Individual files

With the scripts and methods described in this document you cannot create DCSSs that combine both granularities. This section helps you to decide which granularity is more suitable for your needs.

Sharing directories

You can share data at the granularity of entire directories as illustrated in Figure 5 on page 6.

Advantages: Sharing entire directories allows you to share a large number of files with only a few bind-mount operations, and therefore with only a little overhead in memory consumption.

Disadvantages: Every shared directory and its subdirectories are read-only. You cannot share read-only files in directories that also contain files that are written to.

Sharing files at the granularity of entire directories is advantageous if:
- You want to share a large number of small files.
- You do not want to share files in directories that are written to.

Sharing individual files

The left side of Figure 6 shows a Linux file system with the corresponding DASD volume where it resides as a file system image. Directories `/c` and `/d` are shown with a small number of the files they contain. Some of these files are to be shared.

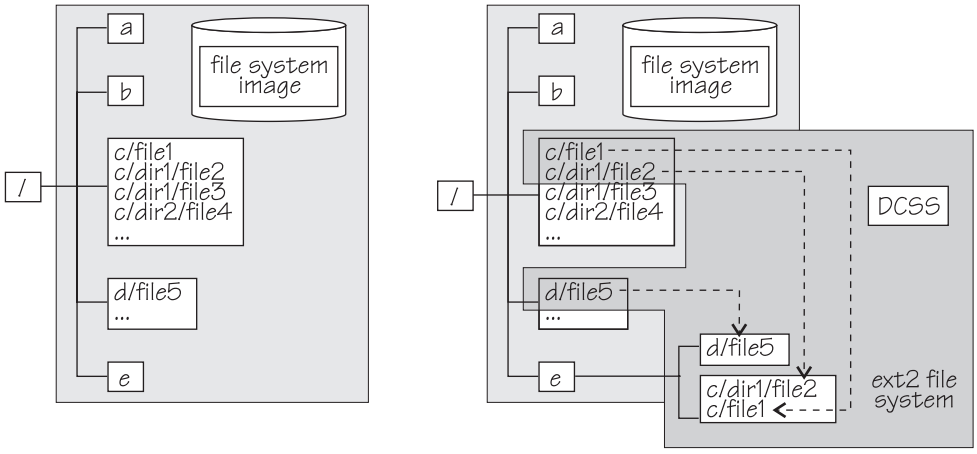

Figure 6. ext2 with the xip mount option over-mounting individual files

On the right side, a DCSS contains individual files, `/c/file1`, `/c/dir1/file2`, and `/d/file5`. Mounting the ext2 file system from this DCSS over-mounts only these individual files. With the DCSS mounted on a mount point e, calls to these files are

redirected to `/e/c/file1`, `/e/c/dir1/file2`, and `e/d/file5`, respectively. Calls to other files in directories c and d are not affected.

Advantages: Sharing individual files allows you to be very specific about what you want to share and what you do not want to share. Directories with shared read-only files are not limited to reading, new files can be created in them at any time, and files that have not been selected to be shared can be written to.

Disadvantages: Sharing files at the granularity of individual files requires a bind-mount operation for each shared file. Every bind-mount operation consumes up to 750 byte of storage for each guest that shares the DCSS. For a large number of shared files, there can be considerable storage consumption on account of bind-mount operations.

Sharing files at the granularity of individual files is advantageous if:

- You want to share a small number of files.
- You want to share files that reside in directories that are written to.
- You want to share files that are spread over a large number of directories, while numerous files in these directories do not need to be shared.

Limitations and trade-offs you must be aware of

- DCSSs occupy spool space. When you update a DCSS, multiple versions of the DCSS might coexist. Ensure that your available spool space is at least a multiple of the DCSS size.
- The address range where the DCSS is located is the same for all guest operating system instances and needs to be set when the DCSS is created.
- The address range must not overlap with the guest storage (perceived as physical memory by Linux) of any operating system instance that accesses it.
- Linux needs memory management data structures to access data on the DCSS. These data structures occupy 13 MB (31-bit Linux kernel) or 26 MB (64-bit Linux kernel) per GB DCSS. These data structures are subject to z/VM paging and are accessed infrequently, but be aware that DCSS resource consumption increases with size. Sharing files or directories that contain rarely used data can degrade the performance and scalability of a server farm.
- Sharing individual files rather than entire directories can lead to an increase in storage consumption of up to 750 byte for each file. These data structures are subject to z/VM paging and are accessed infrequently, but be aware that resource consumption increases with the number of shared files.
- Some of the tasks you need to perform to set up or update a DCSS include commands that might take considerable time to complete. Be sure that the z/VM watchdog is not active on any Linux guest that you are using to perform these tasks. The watchdog might time out and restart your Linux guest.

Using multiple DCSSs

You can set up multiple DCSSs to be used by the same Linux guest. A Linux guest can concurrently use multiple DCSSs above the storage limit or multiple DCSSs in the first storage gap, but not a combination of the two. To use multiple DCSSs concurrently, you need to ensure that the address ranges of any two used DCSSs do not overlap.

For z/VM 5.4 or later, you can map a DCSS block device to multiple DCSSs that are contiguous in memory. Therefore, the size of a DCSS block device can be much larger than the maximum DCSS size of 2047 MB.

This document describes using a single DCSS only. For using multiple contiguous DCSSs that are mapped by a single DCSS block device, all steps are same except that you have to create multiple DCSSs in “Task 3: Creating the DCSS” on page 16 map them into a single DCSS block device using the command provided by step 2 on page 18 in “Task 4: Copying the shared data to the DCSS” on page 17.

Considerations for sharing files or directories that are not on the root file system

You might want to share a file or directory that is not on the root file system (the file system mounted on /) but on a separate partition. Sharing files or directories that are on file systems other than the root file system is beyond the scope of this document.

If you want to share such files or directories, you need to be able to adapt the steps in this document. In particular, you need to be able to modify the startup scripts of your Linux distribution and adapt your xipinit.sh or xipinit-fw.sh script (see “Task 7: Providing a script to over-mount shared data on startup” on page 23) to ensure that:

- Any file systems with directories or files to be over-mounted are mounted at the beginning of your script, before the bind mount operations.
- The file systems with directories or files to be over-mounted are not mounted again after the DCSS has been mounted.

Page range type considerations

When defining page ranges for a DCSS, you can define them with shared access type SN or SR. Access type SR renders a page range as read-only, SN as “read/write, no data saved”. For details, see the description of the DEFSEG type parameter in *z/VM CP Commands and Utilities Reference*, SC24-6175.

Table 4 compares the merits of SN type ranges versus SR type ranges.

Table 4. SN versus SR type page ranges

SN type	SR type
No persistent copy of the DCSS content is kept by z/VM.	DCSS contents are permanently saved in the z/VM SPOOL space.
Saving and loading the DCSS does not take much time.	Saving and loading a DCSS might take considerable time to complete. While saving or loading is in progress, NSS commands, DCSS commands, or diagnose functions for NSS and DCSS that are issued from any guest operating system of the same z/VM might be delayed.
The DCSS contents are lost when the last user unloads the DCSS, when the system is IPLed, or in the event of a system failure.	The DCSS contents are preserved across IPLs and in the event of a system failure.
The DCSS has to be initialized with its content when it is loaded by the first user.	The DCSS is ready to be loaded by a user at any time.
The DCSS only uses a small amount of z/VM SPOOL space storage for DCSS data structures.	The DCSS consumes considerable z/VM SPOOL space for storing the DCSS contents. You might have to restart z/VM to define sufficient SPOOL space.

The disadvantages of SR type page ranges become more severe as the size of a DCSS increases. In most cases, SR type is preferred for small DCSS while SN type is more suitable for large DCSS.

DCSS options

The z/VM DCSS device driver always saves DCSSs with default properties. Any options that have previously been defined are removed. For example, a DCSS that has been defined with the LOADNSHR operand no longer has this property after being saved through the z/VM DCSS device driver.

To save a DCSS with optional properties, you must unmount the DCSS device, then use the CP DEFSEG and SAVESEG commands to save the DCSS. See "Workaround for saving DCSSs with optional properties" on page 20 for an example.

See *z/VM CP Commands and Utilities Reference*, SC24-6175 for information about DCSS options.

Requirements

To be able to use a DCSS:

- All Linux instances that share a DCSS must be guest operating systems of the same z/VM.
- Your Linux distribution must provide the ext2 file system and must have the xip configuration option compiled into the kernel (kernel configuration parameter CONFIG_EXT2_FS_XIP).
- Your Linux distribution must include the DCSS block device driver, either built-in or as a separate module.

To be able to use a DCSS above 2047 MB:

- You need z/VM 5.4 or later.

Chapter 2. Setting up a DCSS

This section describes what you need to do to put a DCSS in place.

Perform the following tasks to provide a DCSS for your virtual Linux server farm:

1. Plan the DCSS content.
2. Plan the size and location of the DCSS.
3. Create the DCSS.
4. Copy the shared data to the DCSS.
5. Set up the guest for accessing the DCSS.
6. Test the DCSS.
7. Provide a script to over-mount shared data on startup.
8. Activate execute-in-place.

Task 1: Planning the DCSS content

This task helps you to decide what to include in a DCSS.

You might want to use the DCSS to share entire read-only file system directories or to share the read-only files of applications and other individual read-only files.

- If you want to share applications and individual files, proceed with "Identifying individual files to be shared."
- If you want to share entire directories, proceed with "Identifying directories to be shared" on page 14.

You need to plan your DCSS such that it fits into the address space of all guest operating system instances that are going to use it (see "Task 2: Planning the size and location of the DCSS" on page 15).

Identifying individual files to be shared

This section helps you to decide what to include in a DCSS if you want to share the read-only files of applications and other individual files. If you want to share entire directories, skip this section and perform the steps in "Identifying directories to be shared" on page 14 instead.

1. List the applications with the highest memory consumption on your Linux guests. These applications are your candidates for sharing. Issue:

   ```
   # ps -e -ouser,pid,size,args | sort +2n -3
   ```

   Processes with the highest memory consumption are listed last in the command output.

   You get the maximum benefit from sharing applications that are used frequently by a large number of Linux guests.
2. Create an empty directory to serve as a repository for the files to be shared.
3. For each application you consider sharing, find out the space needed to accommodate the read-only application files.
   a. Get a copy of the copylibs.sh script (see "copylibs.sh" on page 32). See "Scripts used for setting up a DCSS," on page 31 for information about where you can obtain a copy of this script.

b. Run copylibs.sh to identify the read-only files of a particular application and to copy them to the directory you created in step 2 on page 11. Issue a command of this form:

```
# copylibs.sh -f <application> -d <destination_directory>
```

The script assumes that the application is in the path. Add the application to the path if necessary.

Example: This command copies the read-only files of the httpd Web server to a directory `/dcss`

```
# copylibs.sh -f httpd -d /dcss
```

Result: copylibs.sh determines the directory where the application resides and copies the application's read-only files to the destination directory. The script replicates the application's subdirectory structure. Symbolic links are resolved and files are copied to a subdirectory that corresponds to their actual location. If the destination directory does not exist, copylibs.sh creates it.

Example: Figure 7 shows a simplified example where an application consists of five files:

- `/usr/bin/abc/symlink` is a symbolic link that points to a read-only file `/lib/read3.so`
- `/usr/bin/abc/write1` is a file that can be written to
- `/usr/bin/abc/read1` is a read-only file
- `/usr/bin/abc/xyz/read2` is a read-only file
- `/usr/bin/abc/xyz/write2` is a file that can be written to

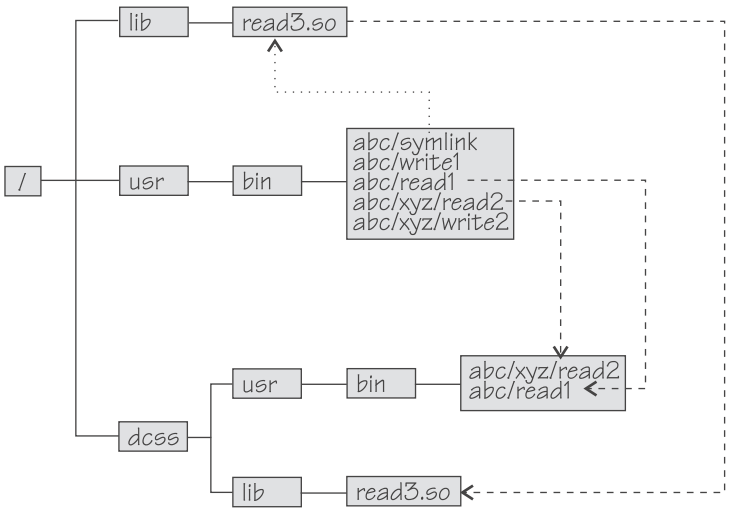

Figure 7. Example: files copied by copylibs.sh

copylibs.sh resolves the symbolic link and copies the read-only files to the destination directory:

- `/dcss/lib/read3.so`
- `/dcss/usr/bin/abc/read1`
- `/dcss/usr/bin/abc/xyz/read2`

c. Determine the space occupied by the copied files:

```
# du -sk <destination_directory>
```

Example:

```
# du -sk /dcss
61761
```

The output shows the space in kilobyte.

Note: The script might not copy all shared objects used by the application. For example, the Apache HTTP Server uses a shared object, `/usr/lib64/apache2` that copylibs.sh does not copy. You can copy such objects as individual files to include them in the DCSS (see step 4).

d. Repeat steps 3b on page 12 and 3c for each application. Find the size of each application from the increment in the output of the `du -sk` command.

4. Copy any individual read-only files or libraries that you know are used frequently. Issue a command like this for each individual file you want to include in the DCSS:

```
# copylibs.sh -f <file> -d <destination_directory>
```

Example: To copy `/usr/lib64/apache2` to `/dcss/usr/lib64/apache2` issue:

```
# copylibs.sh -f /usr/lib64/apache2 -d /dcss
```

5. Find the total size of your DCSS.

```
# du -sk <destination_directory>
```

Example:

```
# du -sk /dcss
882402
```

6. Allow some space for file system metadata. As a conservative estimate, use 4 KB per shared file.

a. Count the files in the DCSS.

Example:

```
# find /dcss | wc -l
21043
```

b. Calculate the additional space requirement.

Example: 21043 × 4 KB = 84172 KB

Tip: Add extra space to allow for future software updates, like security fixes.

Example: The example of the previous steps yields 882402 KB + 84172 KB = 966574 KB as an estimate for the space requirement. Adding extra space for future updates results in a space requirement of: 1 GB = 1024 MB = 1048576 KB.

Hint: DCSSs at the granularity of individual files are often much smaller than 1 GB.

Note: Be aware that each file in your DCSS will be mounted with the **mount --bind** command, and that each mounted file consumes up to 750 bytes of guest storage.

Tip: You can use the destination directory with the files copied in this task as a source for creating the DCSS. However, you cannot remove applications that you have already added. If you decide not to include an application that you have already added, repeat all of step 3 on page 11, omitting the unwanted application.

Proceed with “Task 2: Planning the size and location of the DCSS” on page 15.

Identifying directories to be shared

This section helps you to decide what to include in a DCSS if you want to share entire directories. If you want to share applications and individual files, omit this section and perform the steps in “Identifying individual files to be shared” on page 11 instead.

1. The following list helps you to find directories that are candidates for sharing:
   - Issue the following command to find all regular files with execute permission:

     ```
     # find / -perm -100 -type f
     ```

     Directories that contain any of these files are candidates for sharing.
   - Check the PATH environment variable of the superuser and a normal user. The standard binary directories defined therein are candidates for sharing. Check the directories' contents and include directories that are referred by symbolic links.
   - The major library directories as defined in `/etc/ld.so.conf` are candidates for sharing. For 64-bit Linux, distributions also include the lib64 directories. Check the directories' contents and include directories that are referred by symbolic links.

   The following list helps you to eliminate candidates that are not suitable:
   - Be sure not to share directories that are written to. Within the context of this description, sharing includes subdirectories. You cannot write to a subdirectory of a shared directory.
   - Sharing scripts is ineffective because they are typically interpreted instead of being executed directly. Scripts include, for example, files with extensions .pl, .py, .sh.
   - Do not include the /etc directory.

   You get the most benefit from sharing directories with programs that are frequently used by a large number of the sharing operating system instances.
2. Calculate the space requirements for sharing the directories you have selected.
   a. Issue the following command to find the space occupied by each directory:

      ```
      # du -sk <directories>
      ```

   b. Build the sum of the individual spaces to find the total space occupied by the directories.
   c. Allow some space for file system metadata. As a conservative estimate, use 4 KB per shared file.

      Tip: Add extra space to allow for future software updates, like security fixes.

Example: Assume that the directories to be shared are: `/lib`, `/usr/lib`, `/usr/X11R6/lib`, `/bin`, `/sbin`, `/usr/X11R6/bin`, and `/usr/bin` and that you obtain this space information:

```
# du -sk /lib /usr/lib /usr/X11R6/lib /bin /sbin /usr/X11R6/bin /usr/bin
46596 /lib
562972 /usr/lib
97372 /usr/X11R6/lib
5752 /bin 7768 /sbin
16956 /usr/X11R6/bin
152424 /usr/bin
```

The space used by the directories adds up to 882072 KB. Adding about 10% for metadata and contingencies results in a space requirement of 1 GB = 1024 MB = 1048576 KB.

Task 2: Planning the size and location of the DCSS

This task describes how to determine suitable page frame numbers for the start and end of the DCSS.

1. Determine the maximum size of the DCSS. The maximum size depends on:
   - Whether you are running 31-bit or 64-bit Linux kernels
   - Your z/VM version
   - Whether you want to place your DCSS above the storage of your Linux guests or in a common storage gap
   - The storage size of the largest guest that is to share the DCSS

   Use Table 5 to determine the maximum DCSS size according to your environment. For details on how these formulas are obtained see “DCSS above guest storage” on page 2 and “DCSS in a storage gap” on page 3.

   Table 5. Maximum DCSS size

Kernel	z/VM version	DCSS location	Maximum DCSS size
64-bit	5.4 or later	Either location	2047 MB
	Prior to 5.4	DCSS in a storage gap	1919 MB
		DCSS above guest storage	2047 MB – *<largest guest storage size>*
31-bit		Either location	1960 MB – *<largest guest storage size>*

   Example: Consider a number of Web servers with 64-bit Linux kernels where some have been assigned 256 MB guest storage and others have been assigned 512 MB guest storage. As of z/VM 5.4, the DCSS size is 2047 MB, for a z/VM level prior to 5.4, this example allows the following DCSS size:
   - For a DCSS in a storage gap: 1919 MB (independent of the guest storage sizes)
   - For a DCSS above guest storage: 2047 MB – 512 MB = 1535 MB

   The same size Web servers with 31-bit Linux kernels would allow:
   - 1960 MB – 512 MB = 1448 MB
2. Ensure that the required size (see “Task 1: Planning the DCSS content” on page 11) does not exceed the maximum DCSS size for your environment. If

necessary, remove files or directories from the list of items you want to share. Remove items where you expect the least benefit, that is, where the data is used least frequently.

For 64-bit Linux guest operating systems of z/VM 5.4 or later you do not need to remove files or directories. Instead you can create multiple contiguous DCSSs where the combined size of the contiguous DCSSs is sufficiently large for your requirements. You can map these contiguous DCSSs to a single DCSS block device that is much larger than 2047 MB. In theory, a DCSS block device can be close to 512 GB in size.

3. Decide on a start and end address for the DCSS. The start and end addresses must be on a page boundary (4 KB, on a mainframe) and in hexadecimal notation.
   - For a DCSS in a storage gap choose an address at or above 128 MB.
   - For a DCSS above the guest virtual the start address must be at or above the virtual storage size of the largest guest.

   Examples: The following start and end address provide for a DCSS of 1024 MB.

   Start address: 512 MB => 0x20000000
   End address: 512 MB + 1024 MB - 1 B => 0x5FFFFFFF

   Tip: Round to easily remembered hexadecimal numbers because you will need to use them later in several commands.

   If the DCSS is to be located above the guests storage, this start address limits the guest storage to a maximum of 512 MB. If the DCSS is to be located in a storage gap, guest storage can be allocated below 512 MB and starting from 1024 MB above the DCSS.
4. Calculate the page frame number for the start and end address. You do this by first rounding the address down to the nearest multiple of 4096 and then dividing this number by 4096.

   Hint: In hexadecimal notation this is accomplished by dropping the last three digits.

   Example: Start and end address 0x20000000 and 0x5FFFFFFF result in start and end frame numbers 0x20000 and 0x5FFFF.

Task 3: Creating the DCSS

This task describes how to create the DCSS (see *z/VM CP Commands and Utilities Reference*, SC24-6175 for more information about the commands used in the steps).

Before you start:

- You need to know size of the DCSS you want to share.
- You need to know the start and end address of your DCSS.
- You need privilege class E for your z/VM guest. This guest must be set up to allow a temporary increase of storage using a define store command.

1. Take down Linux and IPL CMS in your guest machine.
2. From CMS, use the **defseg** command to define the DCSS. Issue a command of this form:

```
# defseg <name of the DCSS> <first page number>-<last page number> sr
```

The DCSS name is restricted to 8 characters. The page frame numbers must be in hexadecimal notation. For more details on the **defseg** command, refer to the z/VM CP documentation.

Example:

```
# defseg lnxshare 20000-5ffff sr
```

3. Verify that the DCSS has been defined. Issue:

```
# query nss map
```

The DCSS has been defined correctly if the output shows your segment with the address limits that you have specified.

4. Define your virtual guest storage sufficiently large to cover the entire DCSS. Issue a command of the form:

```
# define stor <value>
```

Specify a value greater than the last page number you used in the **defseg** statement in step 2 on page 16.

Example: To allow for a DCSS with an upper limit of 1536 MB issue:

```
# define stor 1536M
```

Tip: Increasing the size to 2G covers any possible DCSS.

5. Save the DCSS by issuing a command of this form:

```
# saveseg <name of the DCSS>
```

This command can take several minutes to complete.

Example: To save a DCSS named LNXSHARE issue:

```
# saveseg LNXSHARE
```

6. Verify that the DCSS was saved correctly. Issue:

```
# query nss map
```

Result: The save operation has completed successfully if the class (column title: CL) has changed from skeleton (S) to active (A).

7. Log off from your z/VM guest. Next time you IPL an operating system in the guest, the guest storage will be reset to its previous size.

Now you are ready to boot your Linux guest and use the DCSS block device driver to copy the directories or files to the DCSS.

Task 4: Copying the shared data to the DCSS

This tasks describes how to write the shared data to the DCSS.

Before you start:

- You need privilege class E for your z/VM guest virtual machine.

- The maximum definable storage size of your z/VM guest virtual machine must be above the upper limit of the DCSS. Alternatively, suitable authorizations must be in place (see “Accessing a DCSS in exclusive-writable mode” on page 5).
- You need the DCSS block device driver, either as a module or built into the kernel.

1. Ensure that the DCSS block device driver is available. Unless the DCSS block device driver has been built into your kernel or has already been loaded, load the dcssblk module. Issue:

```
# modprobe dcssblk
```

   Result: The device driver has been loaded successfully if `cat /proc/devices` shows an entry “dcssblk”. The number preceding the device driver name is the major number of your DCSS block device.

2. Load the DCSS by adding it to the block device driver. Issue a command of this form:

```
# echo "<name of DCSS>" > /sys/devices/dcssblk/add
```

   With the device nodes of step 3, the first DCSS you add becomes `/dev/dcssblk0`, the second `/dev/dcssblk1`, and so on.

   To create a block device that maps to multiple contiguous DCSSs, issue a command of this form:

```
# echo "<name of DCSS 1>:<name of DCSS 2>:<name of DCSS 3>: ..." > /sys/devices/dcssblk/add
```

   where *<name of DCSS 1>*:*<name of DCSS 2>*:*<name of DCSS 3>*: ... is a list of colon-separated DCSSs that are to be mapped to the DCSS block device.

   Result: Loading a DCSS creates a subdirectory with the name of the DCSS within `/sys/devices/dcssblk`. If you specified a list of DCSSs, the subdirectory name matches the first DCSS in the list. This subdirectory contains several attributes and a symbolic link, block. The symbolic link points to the block device that corresponds to the DCSS. The seglist attribute contains the name of the DCSS or a list of the contiguous DCSSs that have been mapped to the block device. The save and shared attributes are used in the steps that follow.

   Example:

```
# echo "LNXSHARE" > /sys/devices/dcssblk/add
# ls -l /sys/devices/dcssblk/LNXSHARE
total 0
lrwxrwxrwx  1 root root    0 Oct 26 19:29 block -> ../../../block/dcssblk0
-rw-------  1 root root 4096 Oct 26 19:29 save
-r--------  1 root root 4096 Oct 26 19:29 seglist
-rw-------  1 root root 4096 Oct 26 19:29 shared
-rw-r--r--  1 root root 4096 Oct 26 19:29 ueven
```

3. Ensure that there are device nodes for your DCSS block devices. If they are not created for you, for example by udev, create them with the **mknod** command.

   The following lines create 4 nodes:

```
# mknod /dev/dcssblk0 b <major number> 0
# mknod /dev/dcssblk1 b <major number> 1
# mknod /dev/dcssblk2 b <major number> 2
# mknod /dev/dcssblk3 b <major number> 3
```

In the commands, replace *<major number>* with the major number from `/proc/devices`.

The examples for the following steps use the nodes with standard device names as created by the commands in this step. If your distribution provides different device nodes, you can use these nodes instead.

Note: The first DCSS that is loaded on a Linux system is assigned minor number 0, the second minor number 1, and so on. Therefore, for working with a single DCSS a single device node with minor number 0 is sufficient.

4. Change the access mode of the DCSS from “share” (default) to “exclusive-writable”. Issue a command of this form:

```
# echo 0 > /sys/devices/dcssblk/<name of DCSS>/shared
```

Example:

```
# echo 0 > /sys/devices/dcssblk/LNXSHARE/shared
```

5. Create an ext2 file system on the block device that is associated with the DCSS. Use a block size of 4 KB.

Example:

```
# mke2fs -b 4096 /dev/dcssblk0
```

6. Mount the file system in the DCSS on a spare mount point.

Example:

```
# mount -t ext2 -o xip /dev/dcssblk0 /mnt
```

7. Copy the directories or files you want to be shared to the file system on the DCSS. Proceed according to the granularity at which you are sharing data:

   - If you want to share individual read-only files, you will have used copylibs.sh (see “Identifying individual files to be shared” on page 11) to copy the files to a separate destination directory. You can copy the entire destination directory to the file system image:

     Example:

     ```
     # cp -a /dcss/* /mnt
     ```

     Alternatively, you can use copylibs.sh to directly copy the files to the file system on the DCSS. Specify the file system's mount point as the target directory.

     Example:

     ```
     # copylibs.sh -f httpd -d /mnt
     # copylibs.sh -f ...
     ...
     ```

   - If you want to share entire directories, copy these directories to the file system on the DCSS.

     Example:

```
# cp -a /bin /mnt
# cp -a /lib /mnt
# mkdir /mnt/usr
# cp -a /usr/bin /mnt/usr
# cp -a /usr/lib /mnt/usr
```

If your DCSS has been defined with options (see “DCSS options” on page 10) continue with the steps in “Workaround for saving DCSSs with optional properties.” Otherwise complete the remaining steps in this section.

8. Save the DCSS.

   You create a save request by issuing:

   ```
   # echo 1 > /sys/devices/dcssblk/<name of DCSS>/save
   ```

   The save request is performed after the device is unmounted.

   Hint: You can cancel an existing save request by issuing:

   ```
   # echo 0 > /sys/devices/dcssblk/<name of DCSS>/save
   ```

9. Unmount the file system.

   Example:

   ```
   # unmount /mnt
   ```

10. Remove the device. For example, remove the entry in sysfs by issuing:

    ```
    # echo <name of DCSS> > /sys/devices/dcssblk/remove
    ```

Continue with “Task 5: Defining a storage gap” on page 21.

Workaround for saving DCSSs with optional properties

Note: This section applies to DCSSs with special options only. The workaround in this section is error-prone and requires utmost care. Erroneous parameter values for the described CP commands can render a DCSS unusable. Only use this workaround if you really need a DCSS with special options.

Perform the following steps to save a DCSS with optional properties:

1. Unmount the DCSS.

   Example: Enter this command to unmount a DCSS with the device node `/dev/dcssblk0`:

   ```
   # umount /dev/dcssblk0
   ```

2. Use the CP DEFSEG command to newly define the DCSS with the required properties.

   Example: Enter this command to newly define a DCSS, `LNXSHARE`, with the range `80000-9ffff`, segment type `sr`, and the `loadnshr` operand:

   ```
   # vmcp defseg LNXSHARE 80000-9ffff sr loadnshr
   ```

   Note: If your DCSS device maps to multiple DCSSs as defined to z/VM, you must perform this step for each DCSS. Be sure to specify the command

correctly with the correct address ranges and segment types. Incorrect specifications can render the DCSS unusable.

3. Use the CP SAVESEG command to save the DCSS.

   Example: Enter this command to save a DCSS `LNXSHARE`:

   ```
   # vmcp saveseg LNXSHARE
   ```

   Note: If your DCSS device maps to multiple DCSSs as defined to z/VM, you must perform this step for each DCSS. Omitting this step for individual DCSSs can render the DCSS device unusable.

See *z/VM CP Commands and Utilities Reference*, SC24-6175 for details about the DEFSEG and SAVESEG CP commands.

Task 5: Defining a storage gap

This section describes how to set up your guest if you want to locate your DCSS in a storage gap. If you want to locate your DCSS above the virtual storage, skip this section.

If you define multiple storage gaps, be aware that DCSSs can only be located in the storage gap with the lowest address range.

To define a storage gap for your guest, establish a CMS session with the VM guest that you want to set up for accessing the DCSS. Then add a line of this form to your PROFILE.EXEC:

```
DEF STOR CONFIG 0.<gap_start>M <gap_end>M.<guest_storage>M
```

where:

<gap_start>
: defines the lower boundary of the storage gap. The lower boundary must be lower than the start address of the DCSS but not below 128 megabytes.

<gap_end>
: defines the upper boundary of the storage gap. The upper boundary must be higher than the end address of the DCSS. For 64-bit Linux guests the upper boundary can be up to 2047 MB for 31-bit Linux guests it can be up to 1960 MB.

<guest_storage>
: specifies the size of the storage extent above the storage gap.

For more information about the DEF STOR CONFIG command, see *z/VM CP Commands and Utilities Reference*, SC24-6175.

Example: This example defines a storage gap between 500 and 1600 megabytes and sets your guest storage to 750 megabytes (500 megabytes below the storage gap and another 250 above).

```
DEF STOR CONFIG 0.500M 1600M.250M
```

Task 6: Testing the DCSS

This task describes how you can assure that your DCSS has been set up correctly.

Perform the following steps to test the DCSS:

1. Load the DCSS by adding it to the block device driver. Issue a command of this form:

```
# echo "<name of DCSS>" > /sys/devices/dcssblk/add
```

   This command assigns one of the device nodes you created in step 3 of task "Task 4: Copying the shared data to the DCSS" on page 17) to your DCSS.

   To load a block device that maps to multiple contiguous DCSSs, issue a command of this form:

```
# echo "<name of DCSS 1>:<name of DCSS 2>:<name of DCSS 3>: ..." > /sys/devices/dcssblk/add
```

   where *<name of DCSS 1>*:*<name of DCSS 2>*:*<name of DCSS 3>*: ... is a list of colon-separated DCSSs that are to be mapped to the DCSS block device.

2. Mount the ext2 file system read-only with the xip mount option. Use a command of this form:

```
# mount -t ext2 -o ro,xip <device node> <mount point>
```

   This command can take several minutes to complete.

   Example: Assuming that the DCSS has been assigned to the device node `/dev/dcssblk0` in step 1 and you want to mount to a mount point /mnt issue:

```
# mount -t ext2 -o ro,xip /dev/dcssblk0 /mnt
```

3. Issue the following command to verify that the file system has been mounted correctly:

```
# cat /proc/mounts
```

   The file system has been mounted correctly if a line indicates that the mount point is active with the ext2 file system.

4. Verify that all files that you have copied to the mount point are now accessible. Issue:

```
# cd <mount point>
# tar -cvf /dev/null *
```

   where *<mount point>* is your mount point.

If you encounter any problems, check the following:

- Issue the following command from CP to show the DCSS memory range:

```
# query nss
```

- Issue the following command from CP to verify that the DCSS has been loaded by a guest operating system:

```
# query nss users
```

- Issue the following command from CP to display information about the system owned storage for the DCSS:

```
# indicate nss
```

- If you are using a storage gap, verify that the entire DCSS fits within the gap.
- If you have located the DCSS above the guest storage verify that the virtual machine memory is smaller than the start address of the DCSS.
- Verify that the file system size does not exceed the size of the DCSS.

If you have a problem that is not caused by these possible reasons, your DCSS is most probably not set up correctly. Use the CMS **purge** command to remove the DCSS and repeat the tasks "Task 1: Planning the DCSS content" on page 11 through "Task 6: Testing the DCSS" on page 21.

Issue a command like this:

```
# purge nss name <name of the DCSS>
```

where *<name of the DCSS>* is the name of your DCSS.

Example:

```
# purge nss name LNXSHARE
```

Task 7: Providing a script to over-mount shared data on startup

This task provides information about sample scripts that you can use to over-mount files or directories with the content of a DCSS before Linux accesses them.

In this context, over-mounting a directory or file means replacing it with the contents of the DCSS at system startup. Shared items must be over-mounted before being accessed. If an individual file or a file in a shared directory is accessed before being over-mounted, the accessed file might remain in memory, and so waste memory, after the corresponding file or directory is over-mounted.

Proceed according to how you have planned the DCSS in "Task 1: Planning the DCSS content" on page 11:

- If your DCSS is to contain individual files, follow the steps in "Over-mounting individual files."
- If your DCSS is to contain entire directories, follow the steps in "Over-mounting entire directories" on page 24.

Over-mounting individual files

This section applies if you want to over-mount individual files. If you want to over-mount entire directories, skip this section and perform the steps in "Over-mounting entire directories" on page 24 instead.

You can use the xipinit-fw.sh script (see "xipinit-fw.sh" on page 34) to over-mount individual files. See "Scripts used for setting up a DCSS," on page 31 for information about where you can obtain a copy of this script.

This script will have to run before `/sbin/init`. On system startup, xipinit-fw.sh runs instead of `/sbin/init`. First xipinit-fw.sh over-mounts the shared files and then it runs `/sbin/init`.

Perform these steps to tailor the script to your environment:

1. Find the line `ROMOUNT=""` and specify the mount-point for the DCSS.

   Example: `ROMOUNT="/mnt"`
2. Find the line `XIPIMAGE=""` and specify the name of your DCSS.

   Example: `XIPIMAGE="LNXSHARE"`
3. Find the line `RODEV=""` and specify the device node for the DCSS.

   Example: `RODEV="/dev/dcssblk0"`

   This example uses the device nodes of step 3 in task "Task 4: Copying the shared data to the DCSS" on page 17 and assumes that no other DCSS is accessed.
4. Place the tailored script in the `/sbin` directory, name it xipinit-fw.sh, and ensure that the system administrator root is permitted to run it.

Proceed with "Task 8: Activating execute-in-place" on page 25.

Over-mounting entire directories

This section applies if you want to over-mount entire directories. If you want to over-mount individual files, omit this section and perform the steps in "Over-mounting individual files" on page 23 instead.

Directories that are on the root file system need to be over-mounted before running the first program on system startup: `/sbin/init`.

Directories on other file systems (for example, `/usr/bin` if `/usr` is on a separate DASD) need to be over-mounted right after the startup scripts for your kernel have mounted the other file systems. If you want to share directories that are not on the root file system, you need to modify the startup scripts. Because startup scripts are distribution-specific, this document does not describe how to do this.

You can use the xipinit.sh script (see "xipinit.sh" on page 35) to over-mount directories on the root file system (the file system mounted on /). See "Scripts used for setting up a DCSS," on page 31 for information about where you can obtain a copy of this script.

On system startup, xipinit.sh runs instead of `/sbin/init`. First xipinit.sh over-mounts the shared directories and then it runs `/sbin/init`.

Perform these steps to tailor the script to your environment:

1. Find the line `ROMOUNT=""` and specify the mount-point for the DCSS.

   Example: `ROMOUNT="/mnt"`
2. Find the line `XIPIMAGE=""` and specify the name of your DCSS.

   Example: `XIPIMAGE="LNXSHARE"`
3. Find the line `RODEV=""` and specify the device node for the DCSS.

   Example: `RODEV="/dev/dcssblk0"`

   This example uses the device nodes of step 3 in task "Task 4: Copying the shared data to the DCSS" on page 17 and assumes that no other DCSS is accessed.
4. Specify the directories you want to be shared. Find the line that starts with `RODIRS=` and replace the value with a comma separated list of the directories you want to be shared.

Note: Be sure to end the line with a comma character.

5. Place the tailored script in the `/sbin` directory, name it xipinit.sh, and ensure that the system administrator root is permitted to run it.

Task 8: Activating execute-in-place

This task describes how to assure that your shared directories or individual files are over-mounted with the DCSS content at startup.

If all directories or files to be shared are located on the root file system, you just need the xipinit.sh or xipinit-fw.sh script in place as described in "Task 7: Providing a script to over-mount shared data on startup" on page 23.

If you also want to share directories or individual files that are not on your root file system, you have to ensure that all directories to be shared, or the directories containing the individual files to be shared, are mounted to their correct positions before their content is accessed. To do this you need to modify the startup scripts of your Linux distribution.

Perform these steps to activate execute-in-place:

1. Depending on whether you are sharing individual files or entire directories, test your xipinit-fw.sh or xipinit.sh script. Run your script as user root.
   - If you are sharing individual files, issue `/sbin/xipinit-fw.sh`.

     Example:

     ```
     # /sbin/xipinit-fw.sh
     Usage: init 0123456SsQqAaBbCcUu
     ```

   - If you are sharing entire directories, issue `/sbin/xipinit.sh`.

     Example:

     ```
     # /sbin/xipinit.sh
     mounting read-only segment
     binding directory /lib
     binding directory /usr/lib
     binding directory /usr/X11R6/lib
     binding directory /bin
     binding directory /sbin
     binding directory /usr/X11R6/bin
     binding directory /usr/bin
     Usage: init 0123456SsQqAaBbCcUu
     ```

2. Enter `cat /proc/mounts` to confirm that the directories or files to be shared have been over-mounted with the ext2 file system. The output depends on the directories or files you are sharing.

   Examples:

   - The following is part of a sample output when sharing individual files. Depending on the number of shared files, the output might be lengthy:

     ```
     # cat /proc/mounts
     none /mnt ext2 ro,xip 0 0
     none /usr/bin/less ext2 ro,xip 0 0
     none /sbin/fsck ext2 ro,xip 0 0
     none /bin/bash ext2 ro,xip 0 0
     ...
     ```

   - The following is a sample output when sharing entire directories:

```
# cat /proc/mounts
none /mnt ext2 ro,xip 0 0
none /lib ext2 ro,xip 0 0
none /usr/lib ext2 ro,xip 0 0
none /usr/X11R6/lib ext2 ro,xip 0 0
none /bin ext2 ro,xip 0 0
none /sbin ext2 ro,xip 0 0
none /usr/X11R6/bin ext2 ro,xip 0 0
none /usr/bin ext2 ro,xip 0 0
```

Note: Because `/sbin/xipinit.sh` or `/sbin/xipinit-fw.sh` does not add the mounted directories to `/etc/fstab`, the **mount** command does not reflect these mounts properly!

If your `xipinit.sh` or `xipinit-fw.sh` script does not work as intended, revisit "Task 7: Providing a script to over-mount shared data on startup" on page 23.

3. Depending on the granularity at which you are sharing data, add `init=/sbin/xipinit-fw.sh` (individual files) or `init=/sbin/xipinit.sh` (entire directories) to your kernel parameter line (parmfile).
4. Run **zipl** with the new parameter file.
5. Reboot Linux.

Execute-in-place is now active and your Linux instance makes use of the DCSS.

Chapter 3. Making your Linux guests use the DCSS

This task describes these steps you need to perform on each Linux instance that you want to use the DCSS.

Perform these steps to enable a Linux kernel to use a DCSS:

1. Ensure that there are device nodes for DCSS block devices. (see step 3 of task “Task 4: Copying the shared data to the DCSS” on page 17).
2. Depending on the granularity at which you are sharing data, get a copy of the `/sbin/xipinit-fw.sh` or `/sbin/xipinit.sh` script you have adapted in “Task 7: Providing a script to over-mount shared data on startup” on page 23 and activate it on your Linux (see “Task 8: Activating execute-in-place” on page 25).

Chapter 4. Updating the software on a DCSS

This task describes how you can update the data on a DCSS.

Proceed according to the granularity at which you are sharing data and the extend of the intended changes:

- If you want to make minor changes to a DCSS with shared data at the granularity of entire directories, follow the steps in "Updating software on an existing DCSS."
- If you are sharing data at the granularity of individual files or if you are making major updates to a DCSS of any granularity, follow the steps in "Replacing a DCSS" on page 30.

Updating software on an existing DCSS

This task describes how you can perform minor software updates by writing to files on an existing DCSS file system.

Omit this section and perform the steps in "Replacing a DCSS" on page 30 instead:

- If you are intending to make major updates
- If your DCSS contains data at the granularity of individual files
- If your DCSS has been defined with options (see "DCSS options" on page 10)

Before you start:

- You need the DCSS block device driver, either as a module or built into the kernel.
- You need a z/VM guest with Class E user privileges.
- The maximum definable storage size of your z/VM guest virtual machine must be above the upper limit of the DCSS. Alternatively, suitable authorizations must be in place (see "Accessing a DCSS in exclusive-writable mode" on page 5).
- The updated file system image must fit into the existing DCSS.

To update data on an existing DCSS, perform the following steps:

1. Restart your Linux guests without activating execute-in-place. For each guest that shares the DCSS:
   a. Remove the "init=" parmeter from the kernel parameter line (parmfile).
   b. Run **zipl** with the new parameter file.
   c. Boot Linux using the new boot configuration.
2. Perform the required software update on each guest that shares the DCSS.
   Perform the updates according to your distribution documentation, as you would on any platform. Be sure to make the *same updates* on each guest. For example, install or update the same packages with the same version. Do not continue until all updates are completed on all guests that share the DCSS.
3. Run the script update.sh to write the updates to your DCSS. Only run the update.sh once, on only one of your Linux guests.
   a. Get a copy of the update.sh script (see "update.sh" on page 36). See "Scripts used for setting up a DCSS," on page 31 for information about where you can obtain this script.
   b. Run the script as superuser root. The script parses your xipinit.sh file, accesses the DCSS, and updates the DCSS according to your xipinit.sh file.

4. Restart your Linux guests with execute-in-place activated. For each guest that shares the DCSS:
   a. Add `init=/sbin/xipinit.sh` to your kernel parameter line (parmfile).
   b. Run **zipl** with the new parameter file.
   c. Reboot Linux.

Your Linux guests now use the updated DCSS.

Replacing a DCSS

This section describes how to replace your DCSS with a new DCSS that provides the updated software. You can use this update method for any DCSS.

Tip: If you are making minor updates to a DCSS that contains shared data at the granularity of entire directories, you can also proceed as described in “Updating software on an existing DCSS” on page 29 instead of replacing the DCSS.

To replace a DCSS, perform the following steps:

1. Restart your Linux guests without activating execute-in-place. For each guest that shares the DCSS:
   a. Remove the “init=” parmeter from the kernel parameter line (parmfile).
   b. Run **zipl** with the new parameter file.
   c. Reboot Linux using the changed boot configuration.
2. Perform the required software update on each guest that shares the DCSS.

   Perform the updates according to your distribution documentation, as you would on any platform. Be sure to make the *same updates* on each guest. For example, install or update the same packages with the same version. Do not continue until all updates are completed on all guests that share the DCSS.
3. Create a new DCSS as described in Chapter 2, “Setting up a DCSS,” on page 11.
4. On every guest, change the value of XIPIMAGE="" in your /sbin/xipinit.sh or /sbin/xipinit-fw.sh script, to the name of the new DCSS.
5. Restart your Linux guests with execute-in-place activated. For each guest that shares the DCSS:
   a. Add an “init=” parameter to your kernel parameter line (parmfile). The value depends on the granularity of the shared data in the DCSS (see “Granularity of shared data” on page 7):

For sharing at directory level add	For sharing at file level add
init=/sbin/xipinit.sh	init=/sbin/xipinit-fw.sh

   b. Run **zipl** with the new parameter file.
   c. Reboot Linux using the changed boot configuration.

Your Linux guests now use the new DCSS.

Appendix. Scripts used for setting up a DCSS

This appendix contains the scripts that are referred to in the main part of this document:

- copylibs.sh
- xipinit-fw.sh
- xipinit.sh
- update.sh

You can download a tarball with copies of these scripts from:

`www.ibm.com/developerworks/linux/linux390/documentation_dev.html`

If you copy any of the scripts from this appendix, confirm that copying and pasting does not change any characters.

copylibs.sh

You need copylibs.sh for “Task 1: Planning the DCSS content” on page 11 if you want to share the read-only parts of applications and individual files.

```
#!/bin/bash
# copylibs.sh, Version 3
#
# (C) Copyright IBM Corp. 2005
#
#
# copies the binary and the libraries used to destination directory
# usage: ./copylibs.sh -f <executable|file> -d <destination directory>
#
#FAKE=echo


function getlibs {
 for i in `ldd $FILE | awk '{print $3}'`
 do
    echo $i
    if [[ -h $i ]]
    then
       echo $i is a link
       #LINKPATH=${i%/*}
       FILELINKEDTO=$(readlink -f $i)
              #FILELINKEDTOPATH=${FILELINKEDTO%/*}
       echo $FILELINKEDTO
       $FAKE cp -a --parent ${FILELINKEDTO} ${DESTINATION}
    elif [[ -e $i ]]
    then
       $FAKE cp -a --parent $i ${DESTINATION}
    fi
 done

 echo NOTE: Libraries which were loaded with libdl will not be copied.
 echo NOTE: be sure that you copy these extra.
}

###
###
# here the script starts
###
###

if [[ $# -ne 4 ]]
then
 echo "Usage: ./copylibs.sh -f <executable|file> -d <destination directory>"
 exit 1
fi
```

Figure 8. copylibs.sh (1/2)

```
while getopts :f::d: OPT
do
   case $OPT in
       f )    if [[ ! $FILE && ${OPTARG:0:1} != '-' ]]
              then
                 echo "option f is set with $OPTARG"
                 FILE=$OPTARG
                 echo $FILE
       else
          echo error for option -f please check
          exit 1
       fi
       ;;
       d )    if [[ ! $DESTINATION && ${OPTARG:0:1} != '-' ]]
              then
          echo "option d is set with $OPTARG"
                 DESTINATION=$OPTARG
   echo $DESTINATION
       else
          echo error for option -d please check
   exit 1
       fi
   ;;
       :)     echo "no option set" ;exit 1 ;;
   esac
done

#here check if file (full qualified) exists
if [[ -e $FILE ]]
then
   if [[ $(file $FILE|grep ELF > /dev/null) -eq 0 ]]
   then
      getlibs
      # a dynamic shared object, so get all other libs needed
   fi
   #copy file itself first check if file is a symlink
   if [[ -h $FILE ]]
   then
      FILE=$(readlink -f $FILE)
   fi
   $FAKE cp -a --parent $FILE $DESTINATION
elif [[ ! -e $FILE ]]
then
   #check if file is an executable maybe found in PATH
   FILE=$(which $FILE)
   if [[ $? -ne 0 ]]
   then
      echo File not found, exiting...
      exit 1
   else
      $FAKE cp -a --parent $FILE $DESTINATION
      getlibs
   fi
fi
exit
```

Figure 9. copylibs.sh (continued 2/2)

xipinit-fw.sh

You need xipinit-fw.sh for “Task 7: Providing a script to over-mount shared data on startup” on page 23 if you want to share the read-only parts of applications and individual files. If you want to share entire directories use “xipinit.sh” on page 35 instead.

```
#!/bin/bash
#
# xipinit-fw.sh, Version 3
#
# (C) Copyright IBM Corp. 2005


#mount point of xipimage
ROMOUNT=""
#name of xipimage
XIPIMAGE=""
#name of device node
RODEV=""

mount -t sysfs none /sys
if [ ! -e /sys/devices/dcssblk ]; then
  echo "xipinit-fw: loading dcssblk module"
  /sbin/modprobe dcssblk
fi
echo $XIPIMAGE > /sys/devices/dcssblk/add

echo "xipinit-fw: mounting read-only segment"
/bin/mount -t ext2 -o ro,xip $RODEV $ROMOUNT
echo "xipinit-fw: binding files"
for i in $(/usr/bin/find $ROMOUNT/*)
do
  if [[ -f $i ]]; then
       /bin/mount --bind $i ${i#$ROMOUNT}
  fi
done
umount /sys

exec /sbin/init $@
```

Figure 10. xipinit-fw.sh sample script

xipinit.sh

You need xipinit.sh for “Task 7: Providing a script to over-mount shared data on startup” on page 23 if you want to share entire directories. If you want to share individual files use “xipinit-fw.sh” on page 34 instead.

```
#!/bin/sh

#
# xipinit.sh, Version 2
#
# (C) Copyright IBM Corp. 2002,2005
#
# /sbin/xipinit: use read only files from another file system
#
# default options

# internals
#set -v
#FAKE=echo

#mount point of xipimage
ROMOUNT=""
#name of xipimage
XIPIMAGE=""
#name of device node
RODEV=""

RODIRS="/lib,/usr/lib,/usr/X11R6/lib,/bin,/sbin,/usr/X11R6/bin,/usr/bin,"
# make sure it ends with ,
RODIRS="$RODIRS",

mount -t sysfs none /sys
if [ ! -e /sys/devices/dcssblk ]; then
  echo "xipinit: loading dcssblk module"
  /sbin/modprobe dcssblk
fi
echo $XIPIMAGE > /sys/devices/dcssblk/add

# mount ro file system to its mount point
echo "xipinit: mounting read-only segment"
$FAKE mount -t ext2 -o ro,xip "$RODEV" "$ROMOUNT"
# bind mount all ro dirs into rw filesystem
while [ -n "$RODIRS" ] ; do
  dir="${RODIRS%%,*}"
  RODIRS="${RODIRS#*,}"
  test -d "$dir" || continue
  echo "xipinit: binding directory" $dir
  $FAKE mount --bind "$ROMOUNT/$dir" "$dir"
done
umount /sys

# run real init
$FAKE exec /sbin/init "$@"
```

Figure 11. xipinit.sh sample script

update.sh

You need update.sh for “Updating software on an existing DCSS” on page 29. In this task, you make minor updates by writing to an existing DCSS with data at the granularity of entire directories.

```
#!/bin/sh
#
# update.sh, Version 3
#
# (C) Copyright IBM Corp. 2005
#
# This script can be used to mount a DCSS for software updates. Requirements:
# - /sbin/xipinit.sh in with up-to-date RODIRS defined
# - superuser privileges in Linux
# - class E privileges in z/VM

#temporary device node used
TEMPNODE=/tmp/dcssupdate.dev

#uncommend following 2 lines for debugging only
#set -v
#FAKE=echo


parse_xipinit() {
    #looks for xipinit.sh script, and parses its settings
    # save parameters from environment
    _RODEV="$rodev"
    _RODIRS="$rodirs"
    _XIPIMAGE="$xipimage"
    _ROMOUNT="$romount"
    # now parse the script
    echo "reading" /sbin/xipinit.sh
    eval $(cat /sbin/xipinit.sh | grep "RODIRS=" |grep -v 'RODIRS=\"\$' |grep -v "#RODIRS=")
    eval $(cat /sbin/xipinit.sh | grep "RODEV=" |grep -v "#RODEV=")
    eval $(cat /sbin/xipinit.sh | grep "XIPIMAGE=" |grep -v "#XIPIMAGE=")
    eval $(cat /sbin/xipinit.sh | grep "ROMOUNT=" |grep -v "#ROMOUNT=")
    # override parameters with saved environment
    RODEV="${_RODEV:-$RODEV}"
    RODIRS=${_RODIRS:-$RODIRS}
    XIPIMAGE=${_XIPIMAGE:-$XIPIMAGE}
    ROMOUNT=${_ROMOUNT:-$ROMOUNT}
    # make sure it ends with ,
    RODIRS="$RODIRS",
}


check_blockdev() {
    #load dcss block device driver if needed
    test -d /sys/devices/dcssblk || ( echo "loading dcss block device driver" && modprobe dcssblk )
    if [ ! -d /sys/devices/dcssblk ]; then
        echo "error: could not initialize dcss block device driver"
        exit 1
    fi
    echo "dcss block device driver found"
}

load_segment() {
    #load the segment using the block device
    echo "loading segment" $XIPIMAGE
    $FAKE echo $XIPIMAGE >/sys/devices/dcssblk/add
    if [ ! -d /sys/devices/dcssblk/$XIPIMAGE ]; then
        echo "error: failed to load segment"
        exit 1
    fi
    echo "reloading segment" $XIPIMAGE "in nonshared mode"
    $FAKE echo 0 >/sys/devices/dcssblk/$XIPIMAGE/shared
    TEMPVAR0=$(cat /sys/devices/dcssblk/$XIPIMAGE/shared)
    if [ $TEMPVAR0 -ne 0 ]; then
        echo "error: cannot change segment to nonshared mode, maybe you don't have class E privileges?"
        exit 1
    fi
}
```

Figure 12. update.sh (1/3)

```
unload_segment() {
    #unload the segment again
    echo "unloading segment" $XIPIMAGE
    $FAKE echo $XIPIMAGE >/sys/devices/dcssblk/remove
    if [ -d /sys/devices/dcssblk/$XIPIMAGE ]; then
        echo "error: failed to unload segment. is it still busy?"
        exit 1
    fi
}

create_devnode() {
    #create a temporary device node for mounting
    if [ -e $TEMPNODE ]; then
        echo "error: temporary device node" $TEMPNODE "already exists."
        echo "if it still exists from a former run, please delete it and try again"
        exit 1
    fi
    if [ ! -f /sys/devices/dcssblk/$XIPIMAGE/block/dev ]; then
        echo "unexpected-error: cannot find block device directory in sysfs, please report to linux390@de.ibm.com"
        exit 1
    fi
    MAJOR=$(cat /sys/devices/dcssblk/$XIPIMAGE/block/dev |sed -e "s/\:[0-9]\{1,\}//g")
    MINOR=$(cat /sys/devices/dcssblk/$XIPIMAGE/block/dev |sed -e "s/[0-9]\{1,\}\://g")
    mknod $TEMPNODE b $MAJOR $MINOR
    if [ ! -b $TEMPNODE ]; then
        echo "error: cannot not create temporary device node" $TEMPNODE
        exit 1
    fi
}

delete_devnode() {
    #delete temporary device node again
    rm -f $TEMPNODE
    if [ -e $TEMPNODE ]; then
        echo "unexpected-error: cannot remove temporary device node" $TEMPNODE ", please report to linux390@de.ibm.com"
        exit 1
    fi
}

mount_segment() {
    #this mounts the segment on $ROMOUNT
    echo "mounting segment" $XIPIMAGE "on" $ROMOUNT
    $FAKE mount -t ext2 $TEMPNODE $ROMOUNT
    TEMPVAR0=$(cat /proc/mounts |grep $TEMPNODE| wc -l)
    if [ ! 1 -eq $TEMPVAR0 ]; then
        echo "error: cannot mount segment, is either already mounted or operation failed"
        umount $TEMPNODE
        delete_devnode
        unload_segment
        exit 1;
    fi
}

unmount_segment() {
    #unmounts the segment from $ROMOUNT
    echo "unmounting segment" $XIPIMAGE
    $FAKE umount $TEMPNODE
}
```

Figure 13. update.sh (continued 2/3)

```
update_segment() {
    #copies all data from disk to segment
    echo "updating software on segment"
    echo "phase 1: deleting old content"
    TEMPVAR0=$(echo $RODIRS)
    while [ -n "$TEMPVAR0" ]; do
        dir="${TEMPVAR0%%,*}"
        TEMPVAR0="${TEMPVAR0#*,}"
        if [ ! -n "$TEMPVAR0" ]; then
            continue
        fi
        echo "deleting directory" $ROMOUNT/$dir
        if [ ! -d "$ROMOUNT/$dir" ]; then
            echo "error:cannot find directory" $ROMOUNT/$dir
            echo "does not exist on dcss or is not a directory"
            unmount_segment
            delete_devnode
            unload_segment
            echo "cannot recover from earlier error - exit"
            exit 1
        fi
        $FAKE rm -rf $ROMOUNT/$dir
        if [ -d "$ROMOUNT/$dir" ]; then
            echo "error:cannot delete directory" $ROMOUNT/$dir
            unmount_segment
            delete_devnode
            unload_segment
            echo "cannot recover from earlier error - exit"
            exit 1
        fi
    done
    echo "phase 2: copy new content"
    TEMPVAR0=$(echo $RODIRS)
    while [ -n "$TEMPVAR0" ]; do
        dir="${TEMPVAR0%%,*}"
        TEMPVAR0="${TEMPVAR0#*,}"
        if [ ! -n "$TEMPVAR0" ]; then
            continue
        fi
        echo "copying directory" $ROMOUNT/$dir
        mkdir -p $ROMOUNT/$dir
        if [ ! -d "$ROMOUNT/$dir" ]; then
            echo "error:cannot create directory" $ROMOUNT/$dir
            unmount_segment
            delete_devnode
            unload_segment
            echo "cannot recover from earlier error - exit"
            exit 1
        fi
        cp -a /$dir/* $ROMOUNT/$dir
        if [ $? -ne 0 ]; then
            echo "error:cannot copy directory" $ROMOUNT/$dir
            unmount_segment
            delete_devnode
            unload_segment
            echo "cannot recover from earlier error - exit"
            exit 1
        fi
    done
}

save_update () {
    if [ ! -f /sys/devices/dcssblk/$XIPIMAGE/save ]; then
        echo "error:segment or block device driver not loaded"
        exit 1
    fi

    echo 1 >/sys/devices/dcssblk/$XIPIMAGE/save
    TEMPVAR0=$(cat /sys/devices/dcssblk/$XIPIMAGE/save)
    if [ $TEMPVAR0 -eq 0 ]; then
        echo "success: segment saved"
    fi
    if [ $TEMPVAR0 -eq 1 ]; then
        echo "warning: segment scheduled to be saved when it becomes idle, check why it is busy!"
    fi
}


parse_xipinit
check_blockdev
load_segment
create_devnode
mount_segment
update_segment
unmount_segment
delete_devnode
save_update
unload_segment
```

Figure 14. update.sh (continued 3/3)

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information about the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the

names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at

`www.ibm.com/legal/copytrade.shtml`

Intel is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

Readers' Comments — We'd Like to Hear from You

Linux on System z
How to use Execute-in-Place Technology
with Linux on z/VM
March, 2010
Linux Kernel 2.6 – Development stream

Publication No. SC34-2594-01

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.

Submit your comments using one of these channels:

- Send your comments to the address on the reverse side of this form.
- Send your comments via e-mail to: eservdoc@de.ibm.com

If you would like a response from IBM, please fill in the following information:

Name	Address
Company or Organization	
Phone No.	E-mail address

Readers' Comments — We'd Like to Hear from You
SC34-2594-01

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Deutschland Research & Development GmbH
Information Development
Department 3248
Schoenaicher Strasse 220
71032 Boeblingen
Germany

Fold and Tape | **Please do not staple** | Fold and Tape

SC34-2594-01

Cut or Fold
Along Line

SC34-2594-01